Prolog Project: Sudoku Solver

Knowledge representation- Grado en Ingeniería Informática – 2019/2020

Student: Cristian Colavito

Profesor: Jose Carpio Canada

Index

1. INTRODUCION

1.1. Scryer Prolog

To do this project I used Scryer Prolog that is an open source industrial strength production environment that is also a testbed for bleeding edge research in logic and constraint programming, which is itself written in a high-level language.

1.2. What is a Sudoku

Sudoku is a puzzle game designed for a single player, much like a crossword puzzle. The puzzle itself is nothing more than a grid of little boxes called “cells”. They are stacked nine high and nine wide, making 81 cells total. The puzzle comes with some of the cells (usually less than half of them) already filled in, like this:

		6		5	4	9		
1				6			4	2
7				8	9			
	7				5		8	1
	5		3	4		6		
4		2						
	3	4				1		
9			8				5	
			4			3		7

Sudoku cells has to be filled with numbers (1-2-3-4-5-6-7-8-9)

1.3. Rules

The rules of the game are simple: each of the nine blocks must contain all the numbers from 1 to 9 within its squares. Each number can only appear once in a row, column or box(the sub-squares 3x3).

2. SUDOKU SOLVER'S CODE

2.1. How to represent a generic sudoku puzzle

A generic sudoku puzzle may be represented like a list of rows (list of lists) as in the example below:

$$\begin{bmatrix}[X_1,X_2,X_3,X_4,X_5,X_6,X_7,X_8,X_9],\\ [X_{10},X_{11},X_{12},X_{13},X_{14},X_{15},X_{16},X_{17},X_{18}],\\ [X_{19},X_{20},X_{21},X_{22},X_{23},X_{24},X_{25},X_{26},X_{27}],\\ [X_{28},X_{29},X_{30},X_{31},X_{32},X_{33},X_{34},X_{35},X_{36}],\\ [X_{37},X_{38},X_{39},X_{40},X_{41},X_{42},X_{43},X_{44},X_{45}],\\ [X_{46},X_{47},X_{48},X_{49},X_{50},X_{51},X_{52},X_{53},X_{54}],\\ [X_{55},X_{56},X_{57},X_{58},X_{59},X_{60},X_{61},X_{62},X_{63}],\\ [X_{64},X_{65},X_{66},X_{67},X_{68},X_{69},X_{70},X_{71},X_{72}],\\ [X_{73},X_{74},X_{75},X_{76},X_{77},X_{78},X_{79},X_{80},X_{81}]\end{bmatrix}$$

2.2. Sudoku's structure Code:

```
sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs), Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns), maplist(all_distinct, Columns),
    Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
    blocks(As, Bs, Cs), blocks(Ds, Es, Fs), blocks(Gs, Hs, Is).

blocks([], [], []).
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
        all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
        blocks(Ns1, Ns2, Ns3).
```

1 Rows is a list of 9 elements.

Maplist is a predicate using to compact the code.

Append is a predicate to concatenate all elements to the list VS (between 1 and 9).

2 I used the predicate transpose to do the same for the column

3 To define the sub-squares, I named each of the rows and I considered three elements of the rows at the time

4 The first Block is the base case
The second one, if each list starts with 3 elements, so 1 of the sub-squares is complete and all the numbers are distinct.

3. HOW TO SOLVE A SUDOKU PUZZLE

Normally to define a unique sudoku puzzle, I need at least 16 clues, as in the example:

```
?- Rows = [[1,_,_,_,_,_,_,_,_],
           [_,_,2,7,4,_,_,_,_],
           [_,_,_,5,_,_,_,_,4],
           [_,3,_,_,_,_,_,_,_],
           [7,5,_,_,_,_,_,_,_],
           [_,_,_,_,_,9,6,_,_],
           [_,4,_,_,_,6,_,_,_],
           [_,_,_,_,_,_,_,7,1],
           [_,_,_,_,_,1,_,3,_]],
   sudoku(Rows)
   maplist(label, Rows). // to search the concreate solution
```

But I can complete the sudoku without searching anything:

```
?- Rows = [[1,_,_,_,_,_,_,_,_],
           [_,_,2,7,4,_,_,_,_],
           [_,_,_,5,_,_,_,_,4],
           [_,3,_,_,_,_,_,_,_],
           [7,5,_,_,_,_,_,_,_],
           [_,_,_,_,_,9,6,_,_],
           [_,4,_,_,_,6,_,_,_],
           [_,_,_,_,_,_,_,7,1],
           [_,_,_,_,_,1,_,3,_]],
   sudoku(Rows)
   Rows = [[1,8,4,9,6|...]|...].
```

I can do that using the constraint propagation.

I use the predicates ins and all_distinct:

-ins, states the domains of variables
-all_distinct, describe a list of different integers

3.1. Example constraint propagation:

```
?- Vs = [A,B,C], all_distinct(Vs)       //state the domain
   Vs ins 1..3, [B,C] ins 2..3.         //assign to B,C value of 2,3
Vs=[1,B,C],                             //Prolog system deduce that A=1(this is the constraint propagation)
A=1,
B in 2..3,
all_distinct([1,B,C]),
C in 2..3.
```

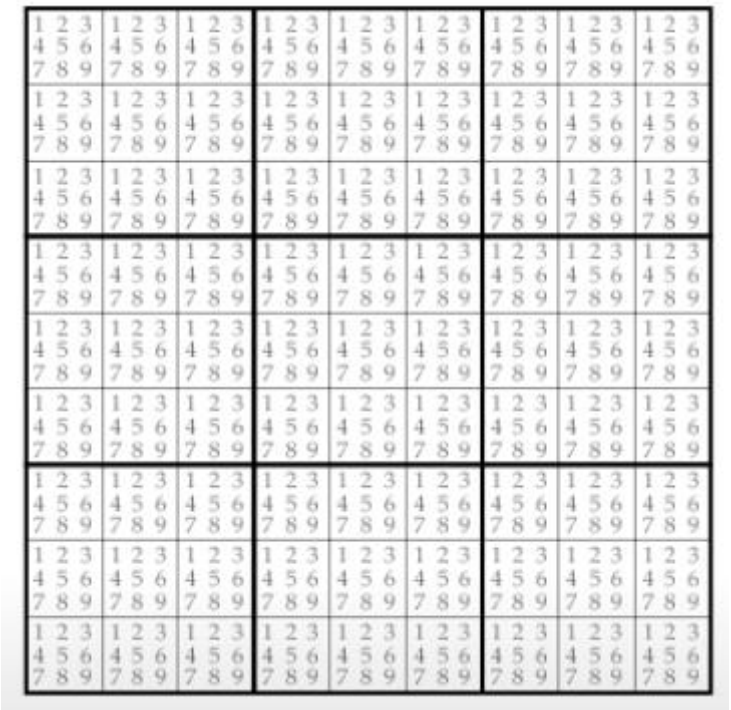
3.2. Consistency Propagation

all_distinct uses a powerful method from **graph theory** to prune the search space.
Prolog automatically detects where will be the solution and also detects which of this sudoku's number will not be the solution, so prolog removes automatically that value from the domain of that cell. In this way prolog obtains the data consistency.

3.2.1. Graph theory

It is the study of graphs, which are mathematical structures used to model pairwise relations between objects.

I can divide each cell in 9 regions (Domain).
the inconsistent value is indicated with a small dot.



When I started the sudoku, this code rules out immediately all the integers signed by the dot.

Advantages about this technique:

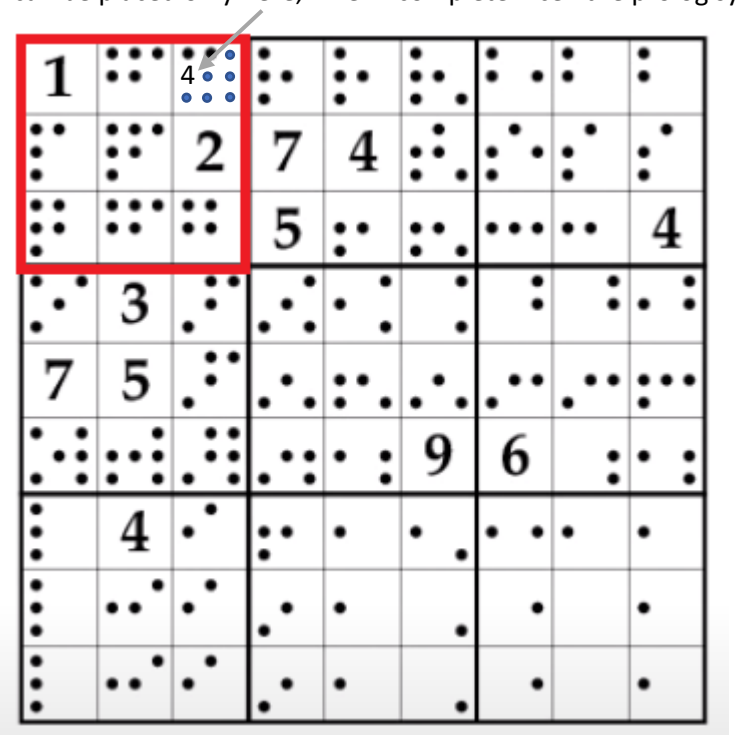
1)I can start with all cells empty, initially each of the integers from 1 to 9 can be place in any cell.

2)you can complete the puzzle without having removed anything that you have already drawn

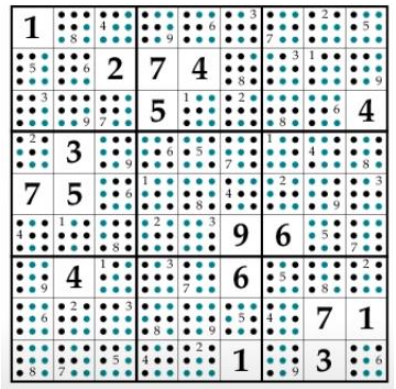
3.3. Example

I considered the past puzzle:

If you focus on the first sub-square you can see that the integer 4 misses only in 1 sub-cell, so 4 can be placed only here, when I complete 1 cell the prolog system, through the consistency propagation, detects that



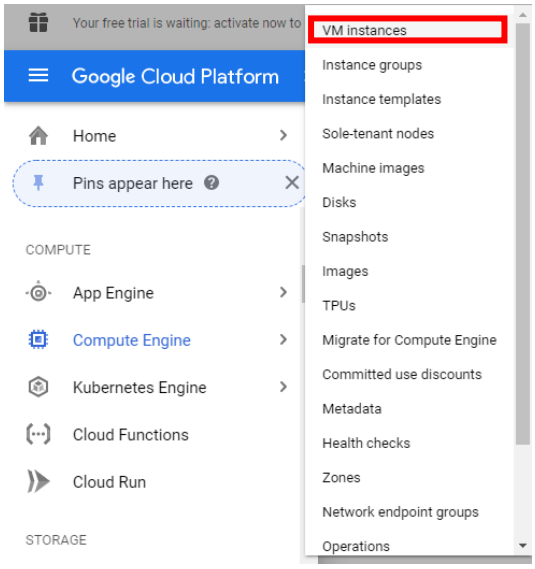
If I do the same for each cell, I can obtain this result:

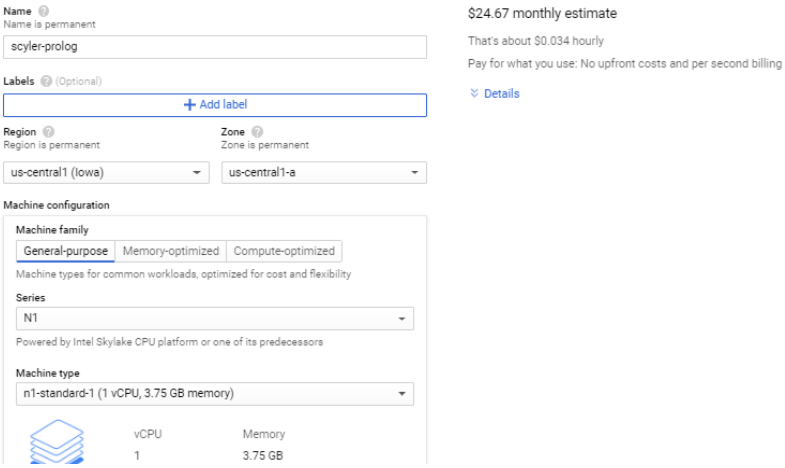


this concrete puzzle was solved by constraint propagation alone, I don't use the research, but in general research is needed to obtain concrete solutions.

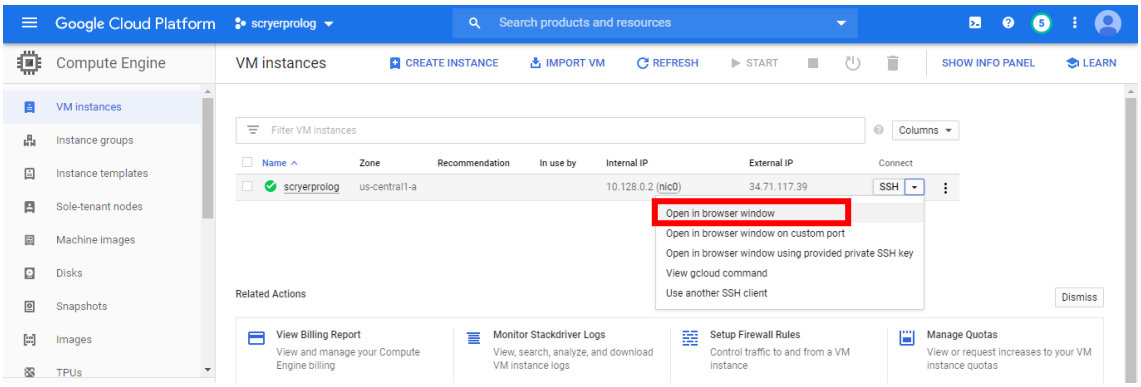
4. HOW TO USE GOOGLE CLOUD PLATFORM TO RUN THIS SUDOKU SOLVER

4.1.1. Create a Virtual Machine & Run





In this panel you can create and set the virtual machine. The costs depend on the type of the machine.



Run the virtual machine in browser window.

4.1.2. Install scryer-prolog

Put in this new window the following codes:

1)
```
cristi_birbo@scryerprolog ~ $ docker run -idt mjt128/scryer-
b36d9e39a7a6dd198f7629a4efd110d54d5c0bee33450f3aa4b4385cb1de
```

2)
```
cristi_birbo@scryerprolog ~ $ docker ps
CONTAINER ID        IMAGE
REATED              STATUS                          PORTS
b36d9e39a7a6        mjt128/scryer-prolog
 seconds ago        Up 6 seconds
460265f8b4e2        mjt128/scryer-prolog
0 seconds ago       Restarting (0) 3 seconds ago
3f16fd59afdf        gcr.io/stackdriver-agents/stackdriver-lo
9 seconds ago       Up 57 seconds
```

This is the <ContainerID>

3)
```
cristi_birbo@scryerprolog ~ $ docker attach b36d9e39a7a6
```

Put here the <ContainerID>

4.1.3 Install nano editor & run

You need to update the apt(Advanced Package Tool) to install nano editor:
`apt-get update`
after this, you can install it:
`apt-get install nano`
run nano with the command: `nano`
write the sudoku code on this link: https://www.metalevel.at/sudoku/sudoku.pl
save like sudoku.pl and that's all.

4.2. Execute the code with scryer prolog

```
root@<ContainerID>:~# scryer-prolog sudoku.pl
```

4.3. Example:

```
?-
Rows = [[1,_,_,_,_,_,_,_,_],
        [_,_,2,7,4,_,_,_,_],
        [_,_,_,5,_,_,_,_,4],
        [_,3,_,_,_,_,_,_,_],
        [7,5,_,_,_,_,_,_,_],
        [_,_,_,_,_,9,6,_,_],
        [_,4,_,_,_,6,_,_,_],
        [_,_,_,_,_,_,_,7,1],
        [_,_,_,_,_,1,_,3,_]],
sudoku(Rows).
```

First solution:

```
Rows=[[1,8,4,9,6,3,7,2,5],[5,6,2,7,4,8,3,1|...],[3,9,7,5,1,2,
8|...],[2,3,9,6,5,7|...],[7,5,6,1,8|...],[4,1,8,2|...],[9,4,1
|...],[6,2|...],[8|...]]
; false.
?-
```

the second solution is false, because it doesn't exist, so it is a unique sudoku puzzle.

5. BIBLIOGRAPHY

- [1] Markus Triska, https://www.metalevel.at/
- [2] What is scryer prolog https://github.com/mthom/scryer-prolog
- [3] What is a sudoku https://www.learn-sudoku.com/what-is-sudoku.html